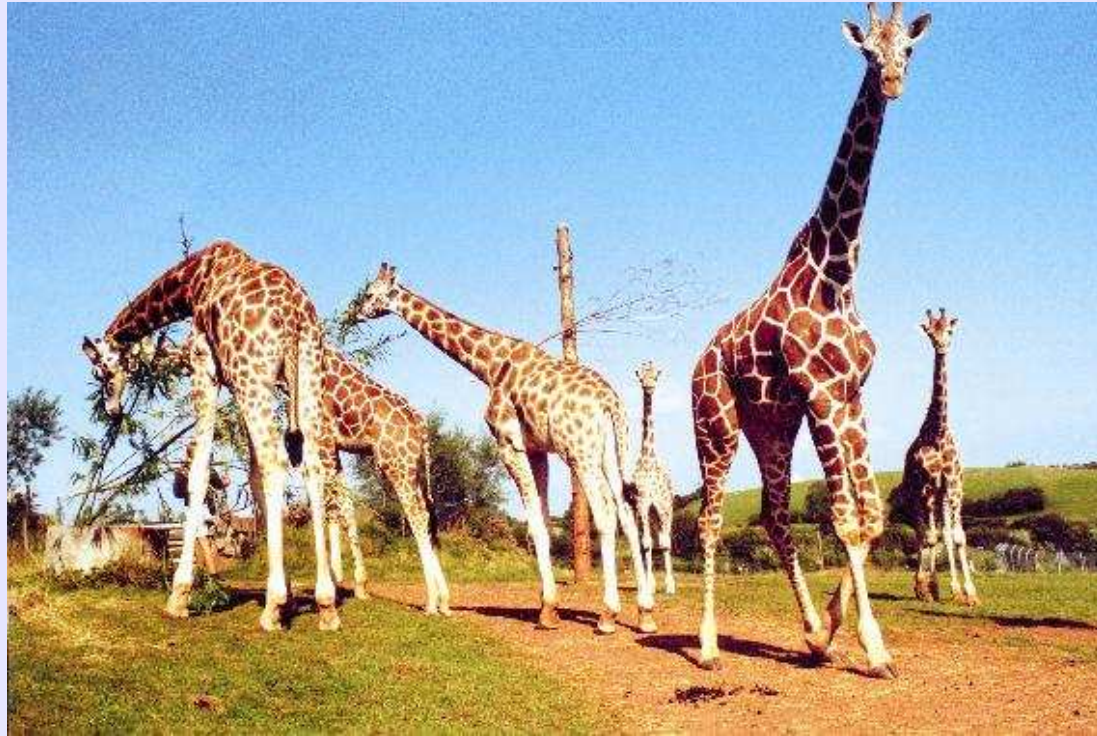


Cache-Oblivious String Dictionaries



Gerth Stølting Brodal

University of Aarhus

Rolf Fagerberg

University of Southern Denmark

SODA 2006, January 22-24, 2006, Miami, Florida

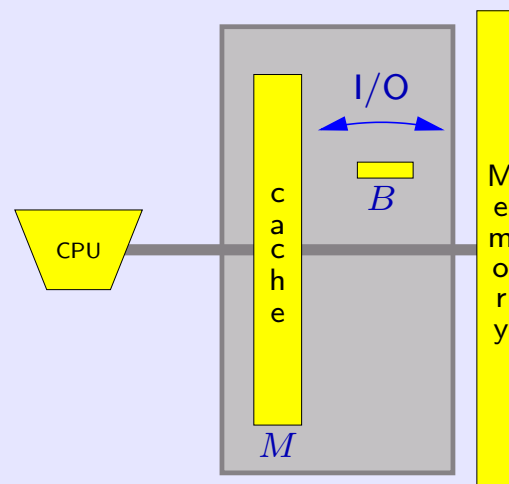
Computational Models



Computational Models

	RAM von Neumann 1946	I/O Model Aggarwal and Vitter 1988	Ideal Cache Model Frigo et al. 1999
Parameters	N	$N M B$	$N M B$
Complexity	# instructions	# I/Os	# I/Os
Properties	Simple	Cache aware: M and B known	Cache oblivious: M and B unknown

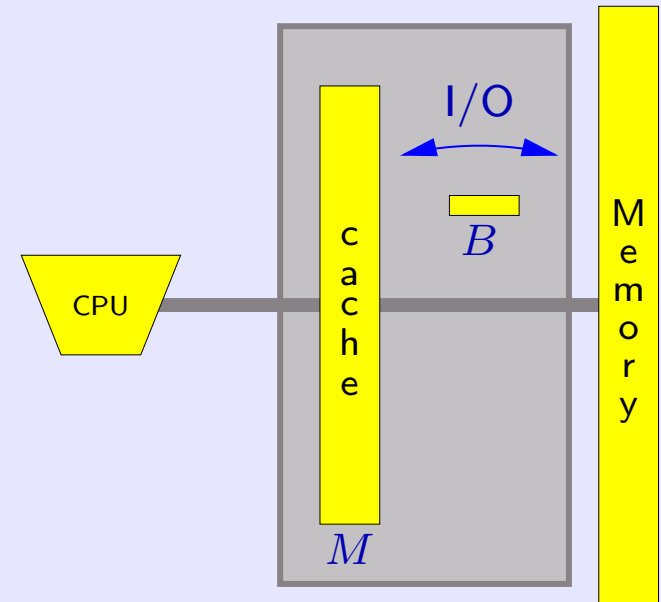
N = problem size
 M = memory size
 B = I/O block size



Ideal Cache Model — no parameters!?

Frigo, Leiserson, Prokop, Ramachandran 1999

- Program with only one memory
- Analyze in the I/O model for
- Optimal off-line cache replacement strategy arbitrary B and M



Advantages

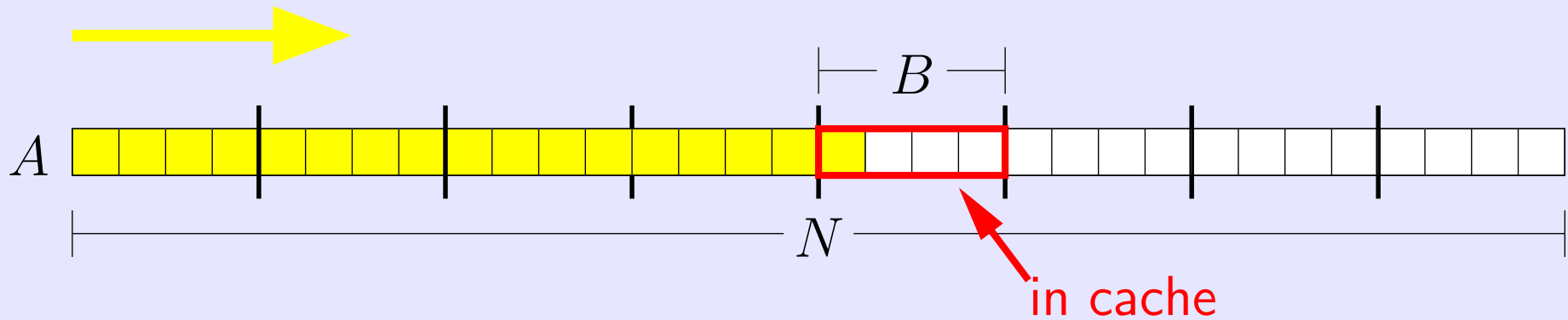
- Optimal on arbitrary level \Rightarrow optimal on **all levels**
- Portability, B and M not hard-wired into algorithm
- DY_nam_ic changing M (and B)



Cache-Oblivious Preliminaries



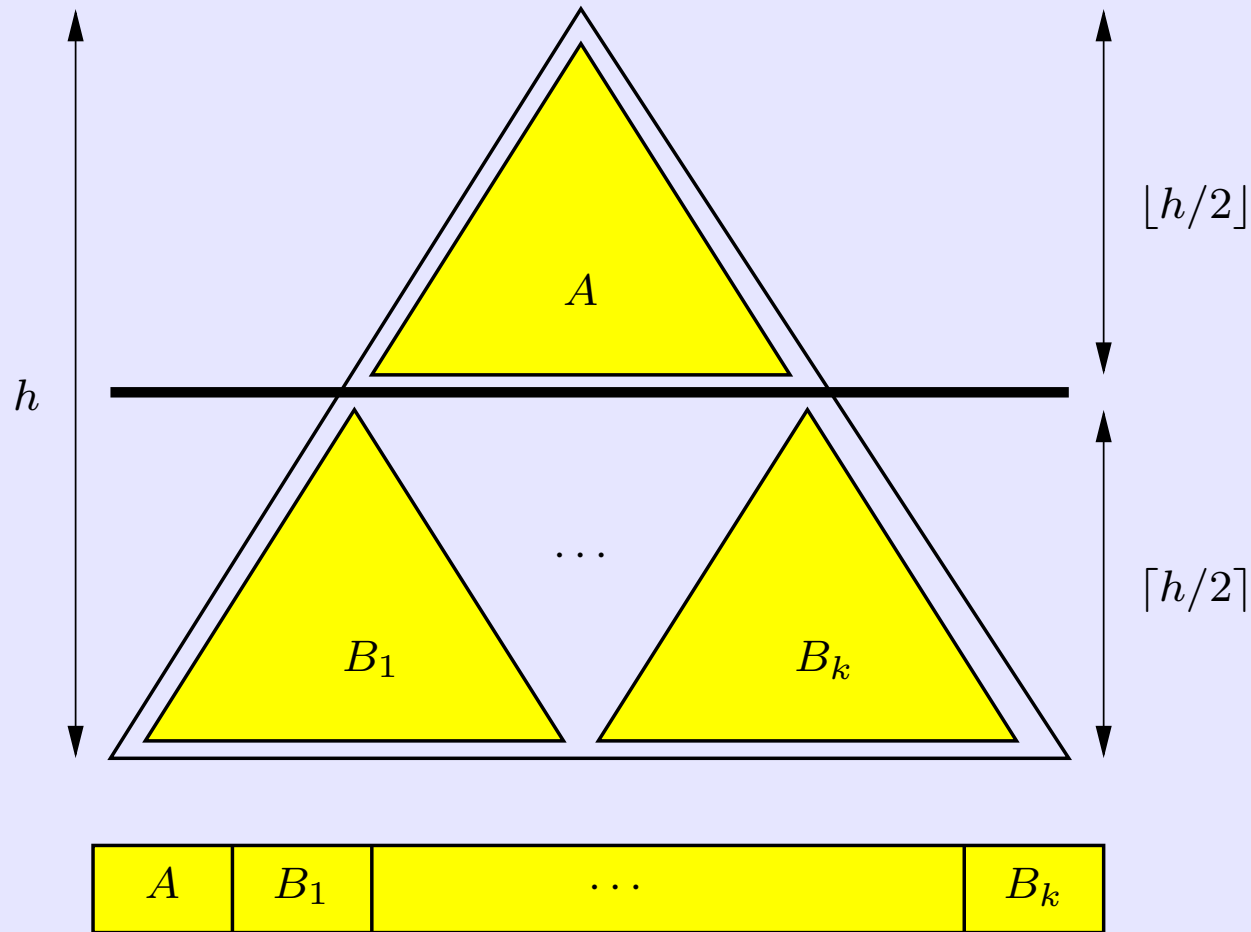
Cache-Oblivious Scanning



$$O\left(\frac{N}{B}\right) \text{ I/Os}$$



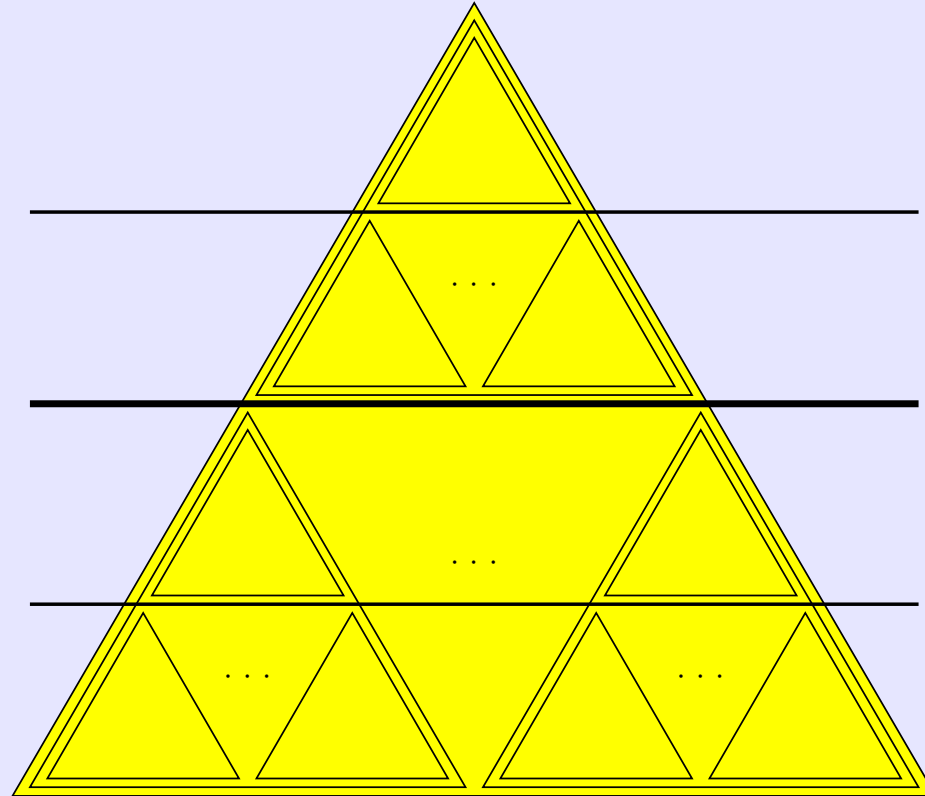
Static Cache-Oblivious B-Tree



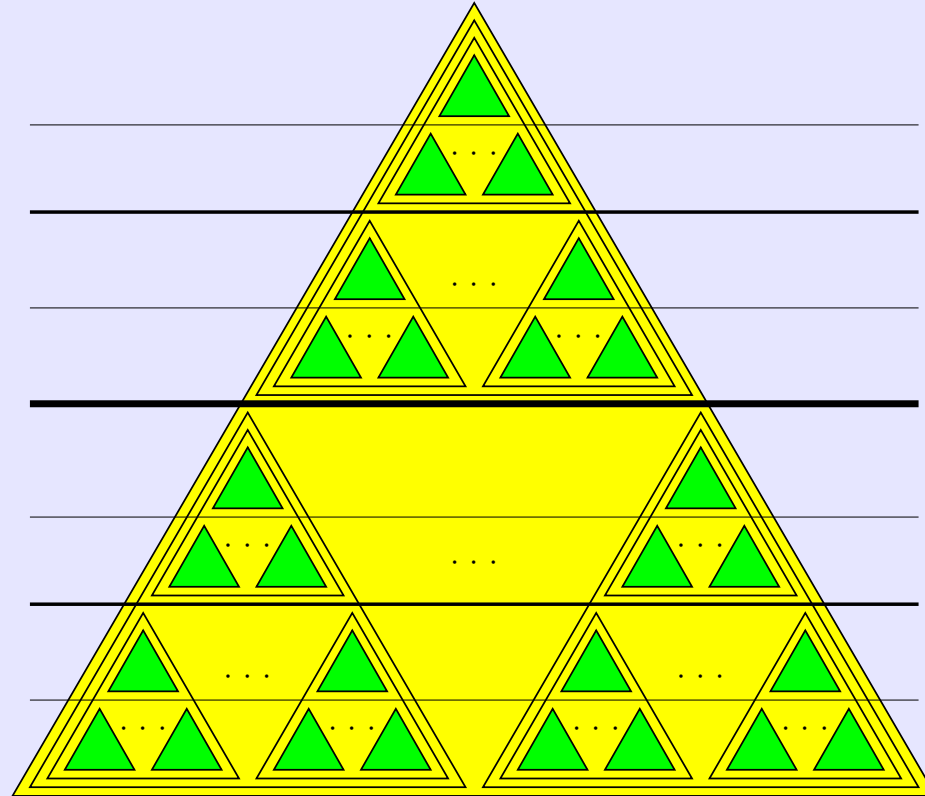
Recursive layout of binary tree \equiv van Emde Boas layout



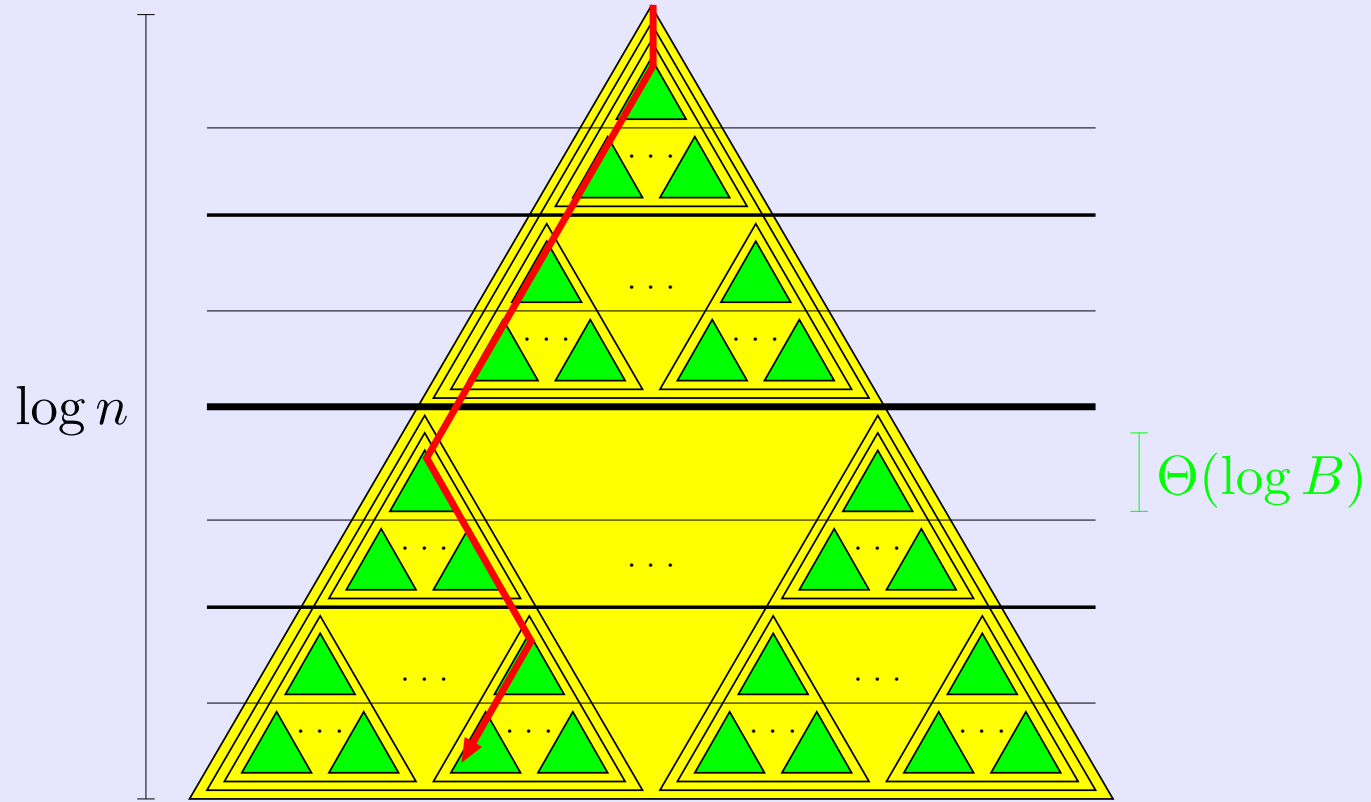
Static Cache-Oblivious B-Tree



Static Cache-Oblivious B-Tree



Static Cache-Oblivious B-Tree



Searches perform $O(\log_B N)$ I/Os



Summary Cache-Oblivious Tools

Scanning : $O(N/B)$

B-tree searching : $O(\log_B N)$

Sorting* : $O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$

* requires a tall cache assumption $M \geq B^{1+\epsilon}$

Frigo, Leiserson, Prokop, Ramachandran 1999

Brodal and Fagerberg 2002, 2003



String Dictionaries



String Dictionaries

	RAM	I/O Model	Ideal Cache Model
Structure	Tries Morrison 1968	String B-trees Ferragina and Grossi 1999	This talk
Query	$O(P)$	$O(P /B + \log_B n)$	$O(P /B + \log_B n)$
Space	$O(N)$	$O(N)$	$O(N)$
	dynamic	dynamic	static, $M \geq B^{2+\epsilon}$

n : number of strings

N : total length of strings

P : query string

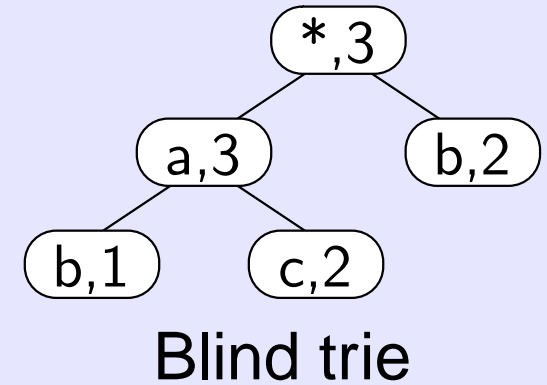
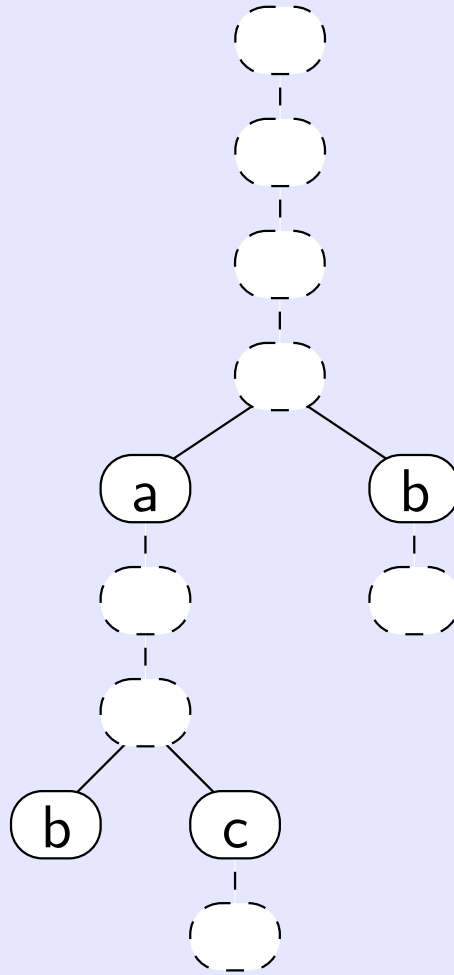
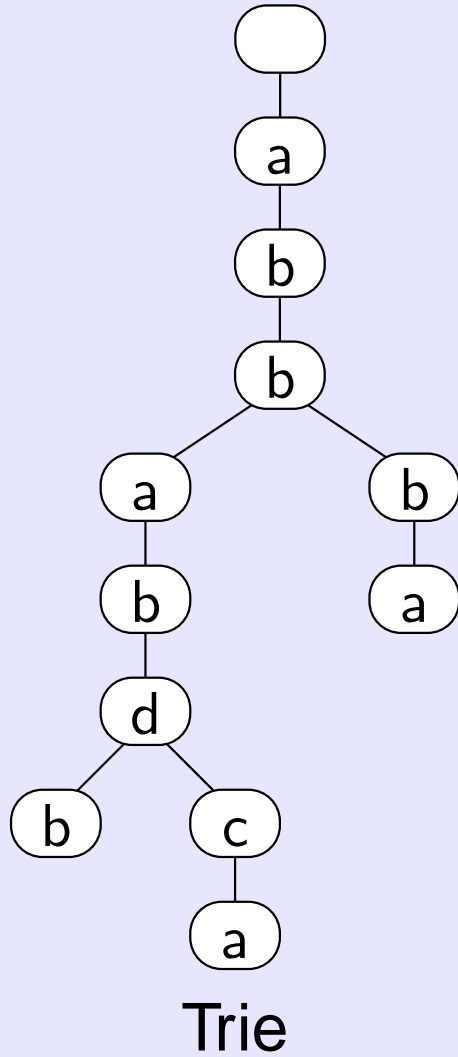


The Trouble Starts...

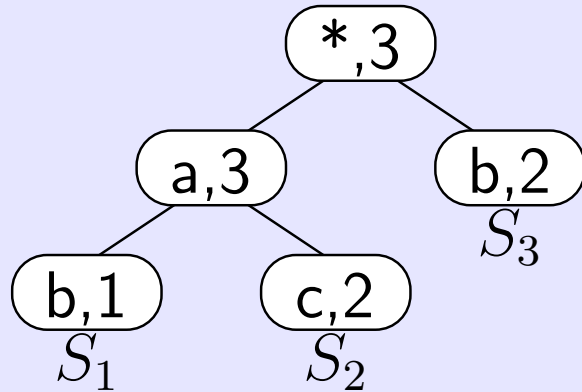
- ÷ Tries cannot be laid out in memory to support top-down searches in $O(\log_B N + |P|/B)$ I/Os Demaine et al. 2004
- ÷ Can construct suffix trees cache-obliviously using $O(\text{Sort}(N))$ I/Os, but cannot search in it efficiently... Farach et al. 2000
- + Cache-aware string B-trees support searches in a set of strings in $O(\log_B n + |P|/B)$ I/Os Ferragina and Grossi 1999



Tries vs Blind Tries



String Dictionary



S_1

a	b	b	a	b	d	b
---	---	---	---	---	---	---

S_2

a	b	b	a	b	d	c	a
---	---	---	---	---	---	---	---

S_3

a	b	b	b	a
---	---	---	---	---

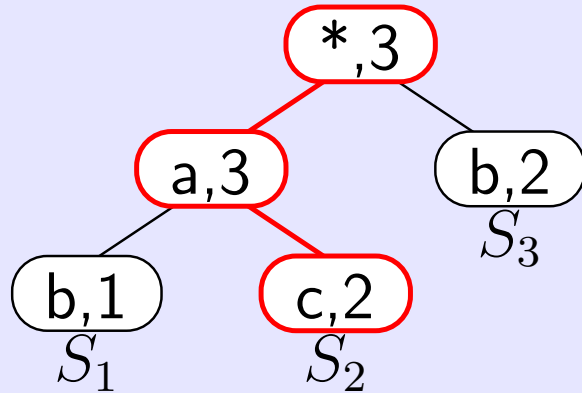
P

a	b	c	a	a	d	c	b
---	---	---	---	---	---	---	---

Queries: Search blind trie + Verify one string



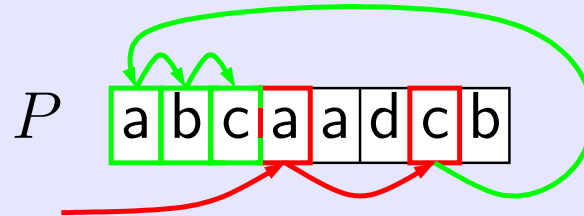
String Dictionary



S_1 a b b a b d b

S_2 a b b a b d c a

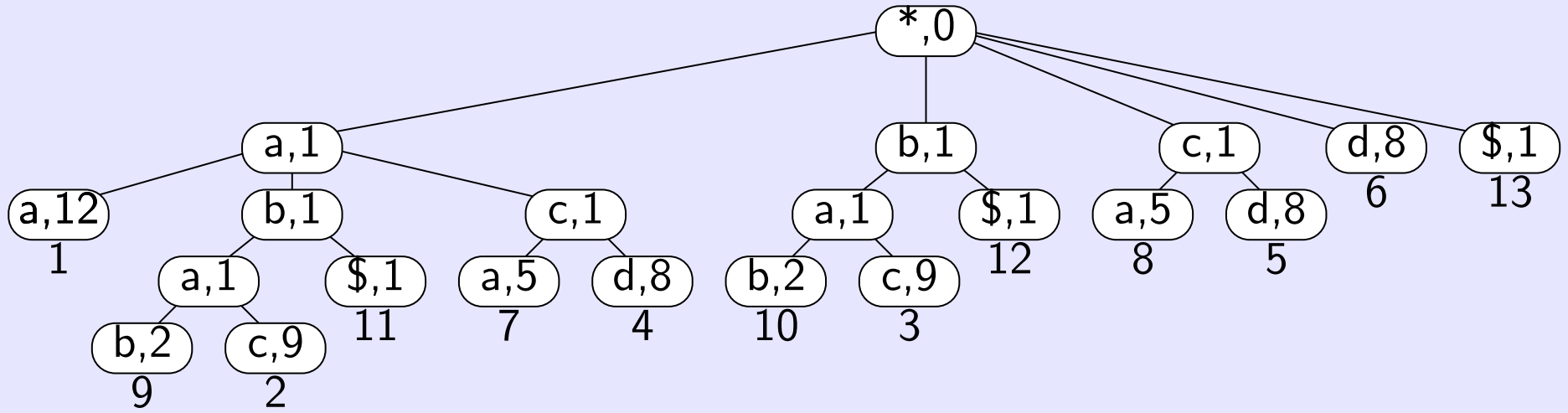
S_3 a b b b a



Queries: Search blind trie + Verify one string



Suffix Tree



T

a	a	b	a	c	d	a	c	a	b	a	b	\$
1	2	...	7	...	13							

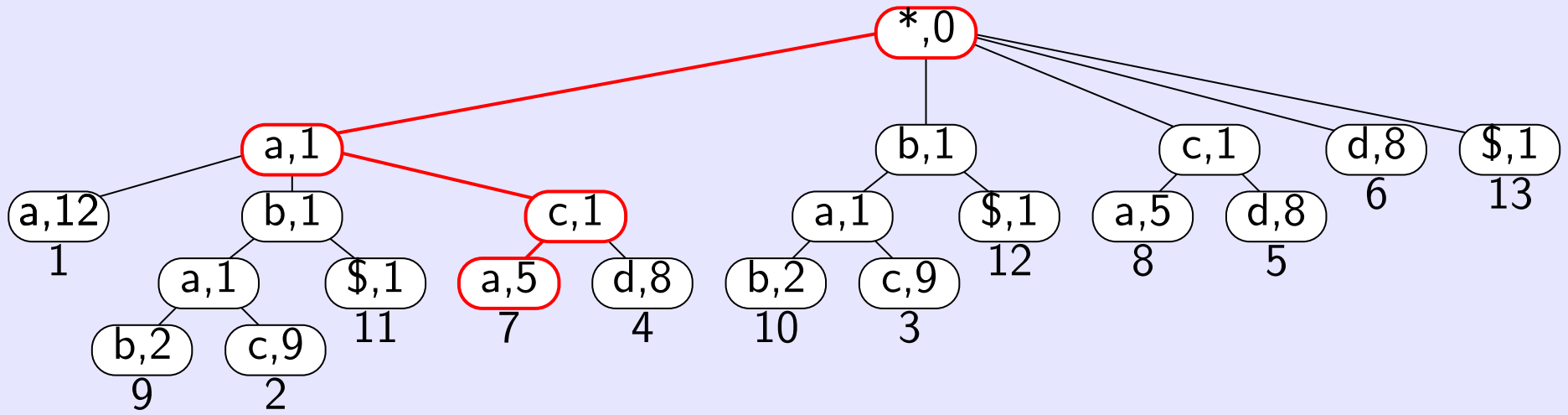
P

a	c	a	d	a
---	---	---	---	---

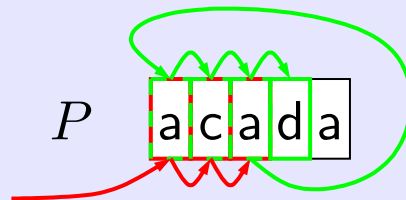
Queries: Search blind trie + Verify one suffix



Suffix Tree



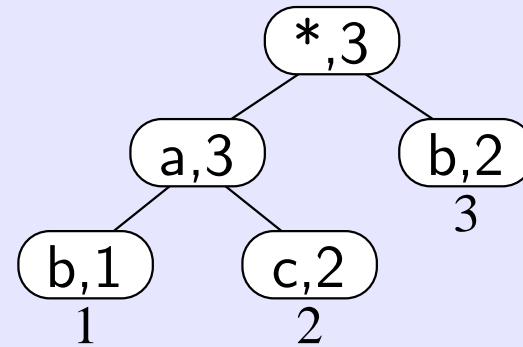
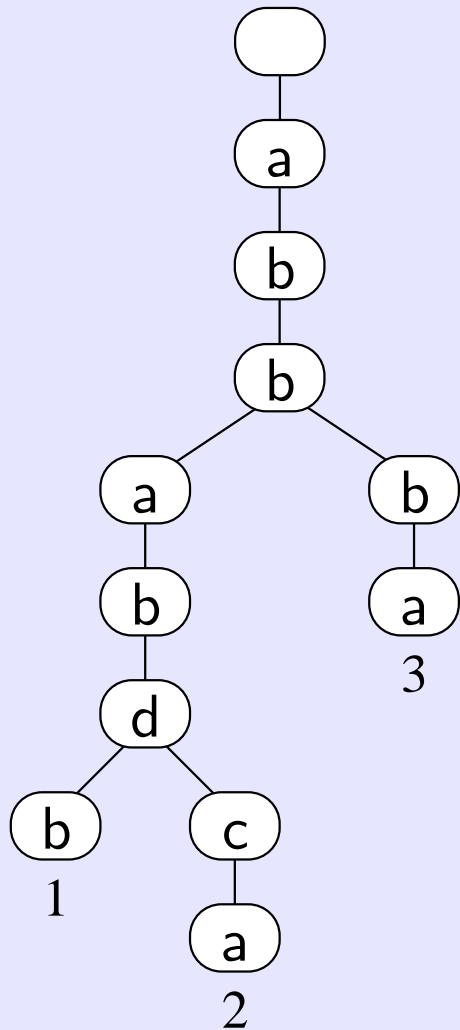
T a a b a c d a c a b a b \$
 1 2 ... 7 ... 13



Queries: Search blind trie + Verify one suffix



Tries



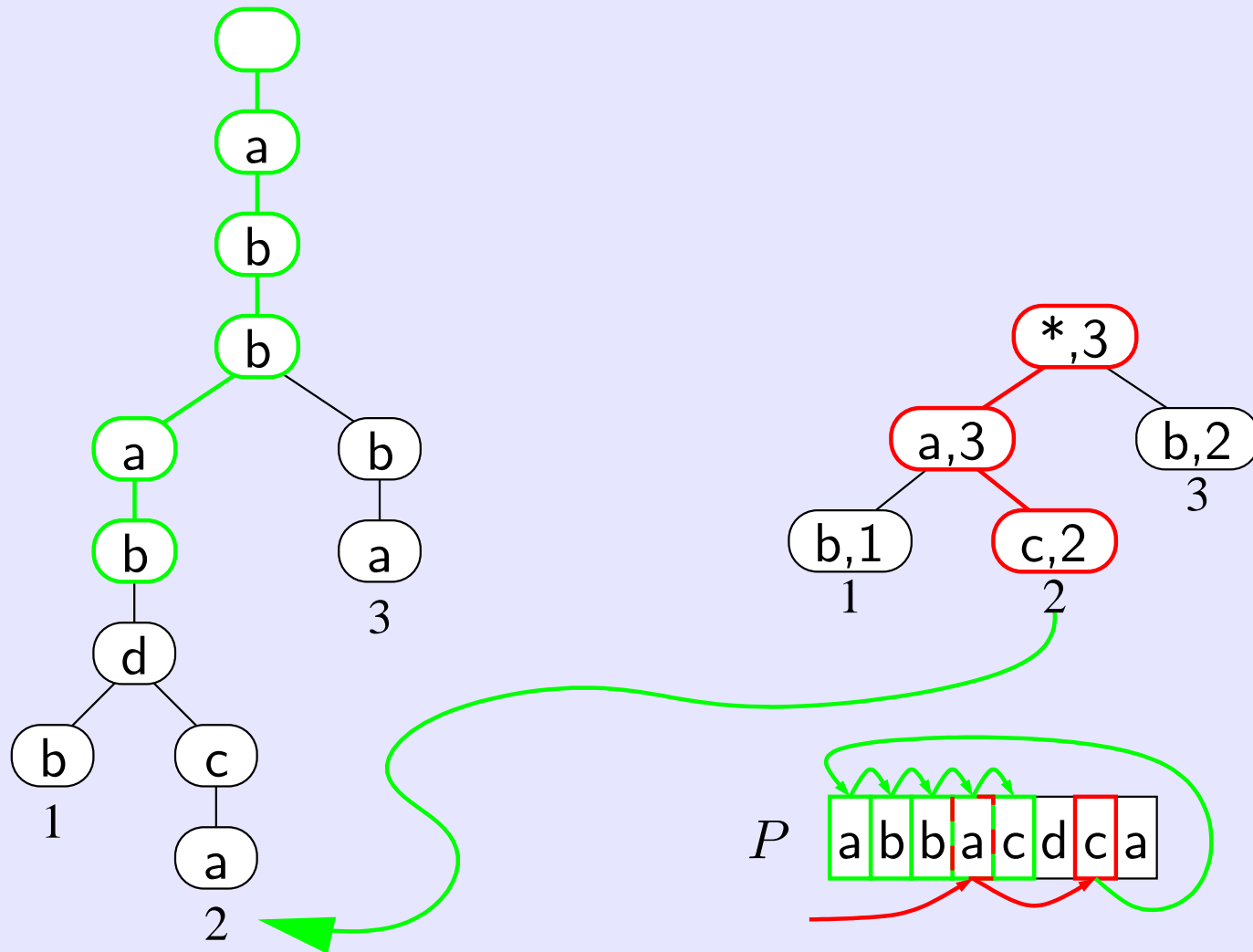
P

a	b	b	a	c	d	c	a
---	---	---	---	---	---	---	---

Queries: Search blind trie + Verify prefix of one path



Tries



Queries: Search blind trie + Verify prefix of one path

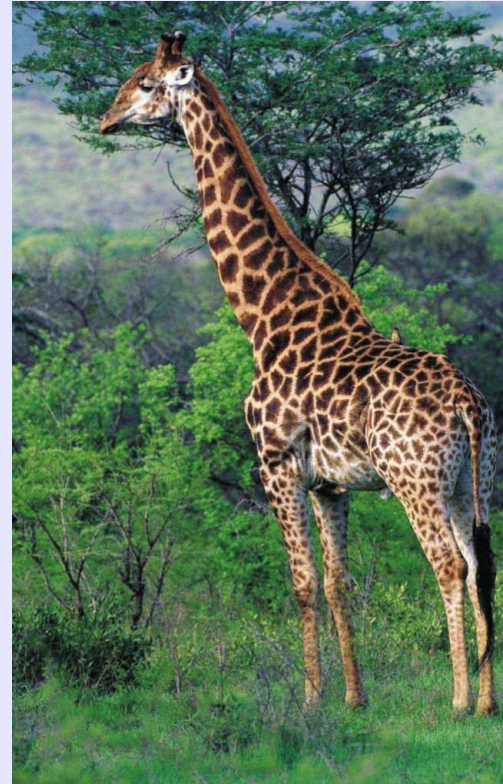
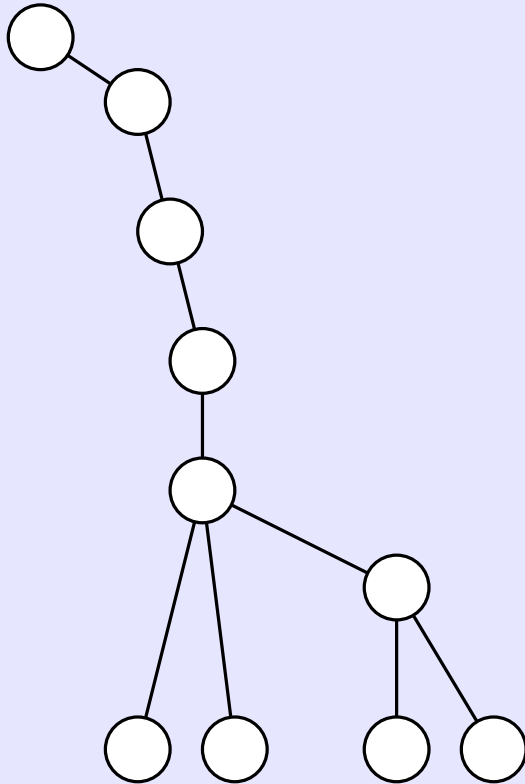


Verifying a Prefix of a Path in a Tree



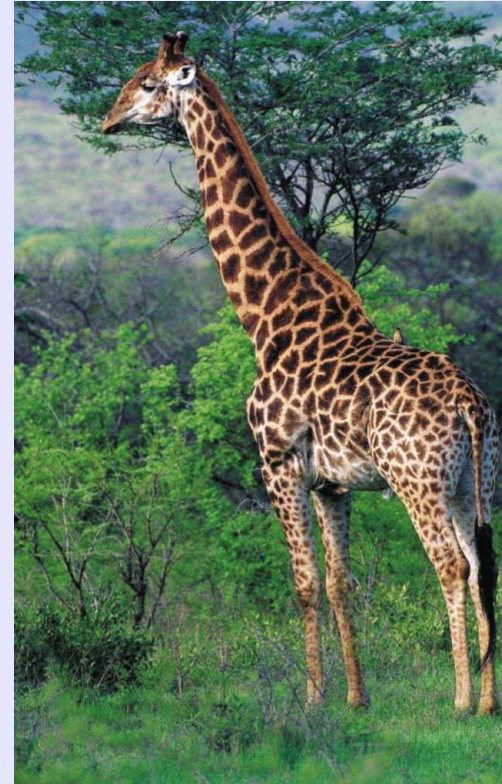
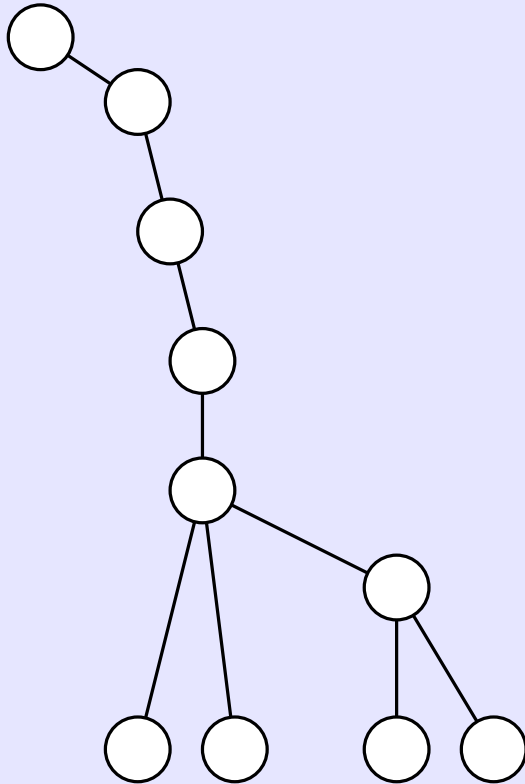
Verifying Paths in Giraffe Trees is Easy

Definition A tree is a **giraffe tree** if all root-to-leaf paths share at least half of the nodes of the tree (long neck)



Verifying Paths in Giraffe Trees is Easy

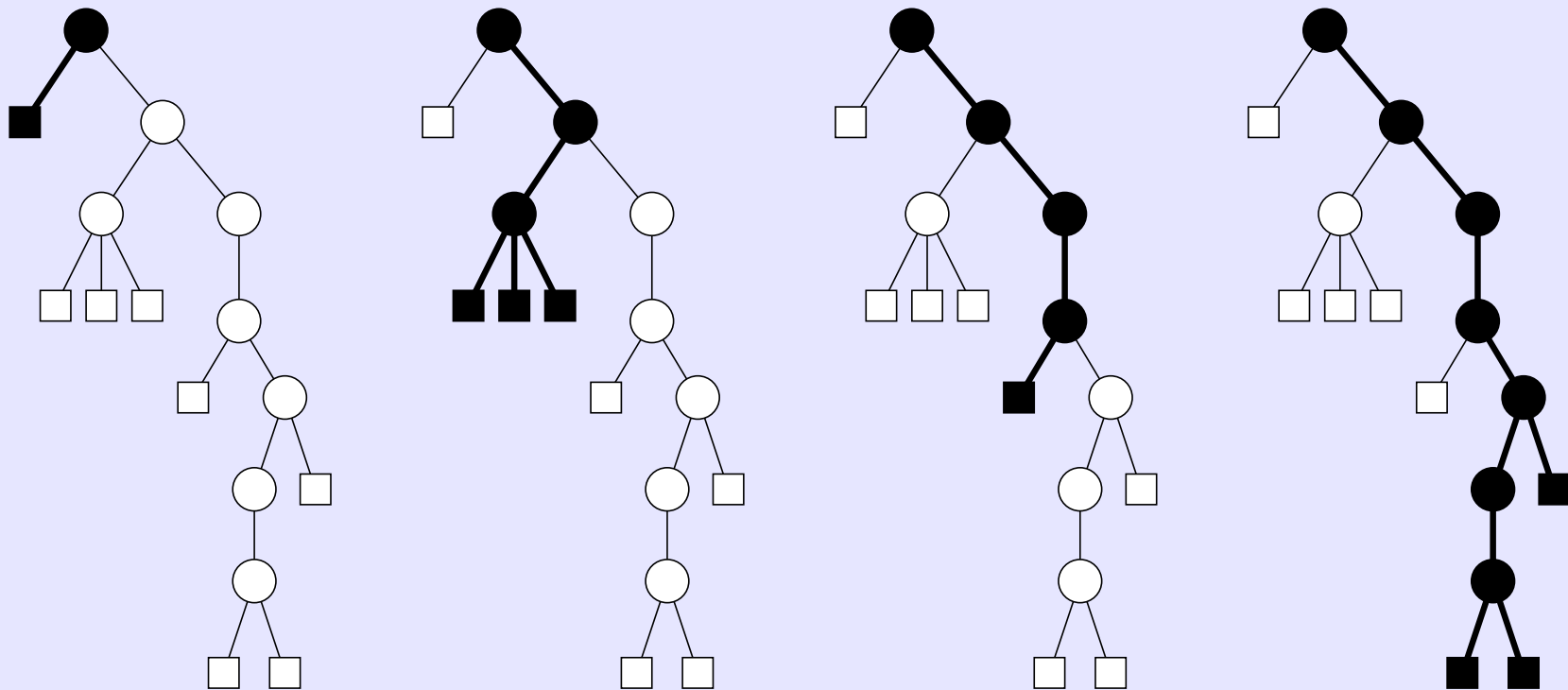
Definition A tree is a **giraffe tree** if all root-to-leaf paths share at least half of the nodes of the tree (long neck)



- A prefix of length p of a path in a giraffe tree using a BFS layout can be traversed in $O(p/B)$ I/Os



Giraffe Cover of a Tree



- Constructed left-to-right using $O(N/B)$ I/Os and space $O(N)$
- A prefix of length p of a path in a known giraffe in BFS layout can be traversed in $O(p/B)$ I/Os



Summary so far...

String dictionary search	}	reduce to blind trie search
Suffix tree search		
Trie search		

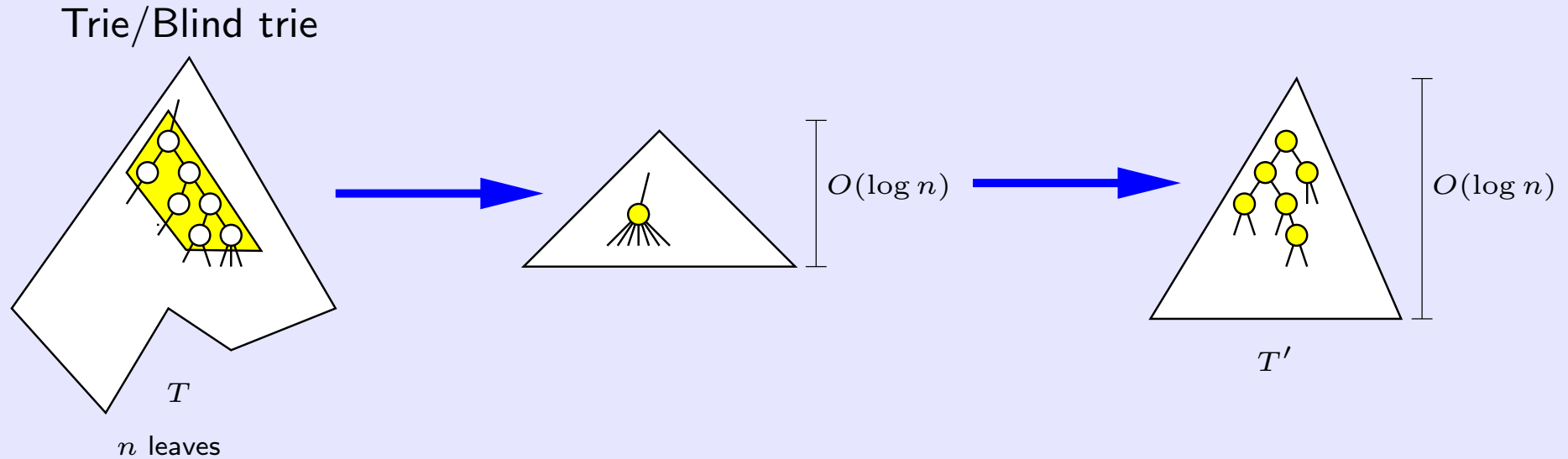
Query : **Blind trie search** + $O\left(1 + \frac{|P|}{B}\right)$ I/Os



Cache-Oblivious (Blind) Tries



Cache-Oblivious (Blind) Tries

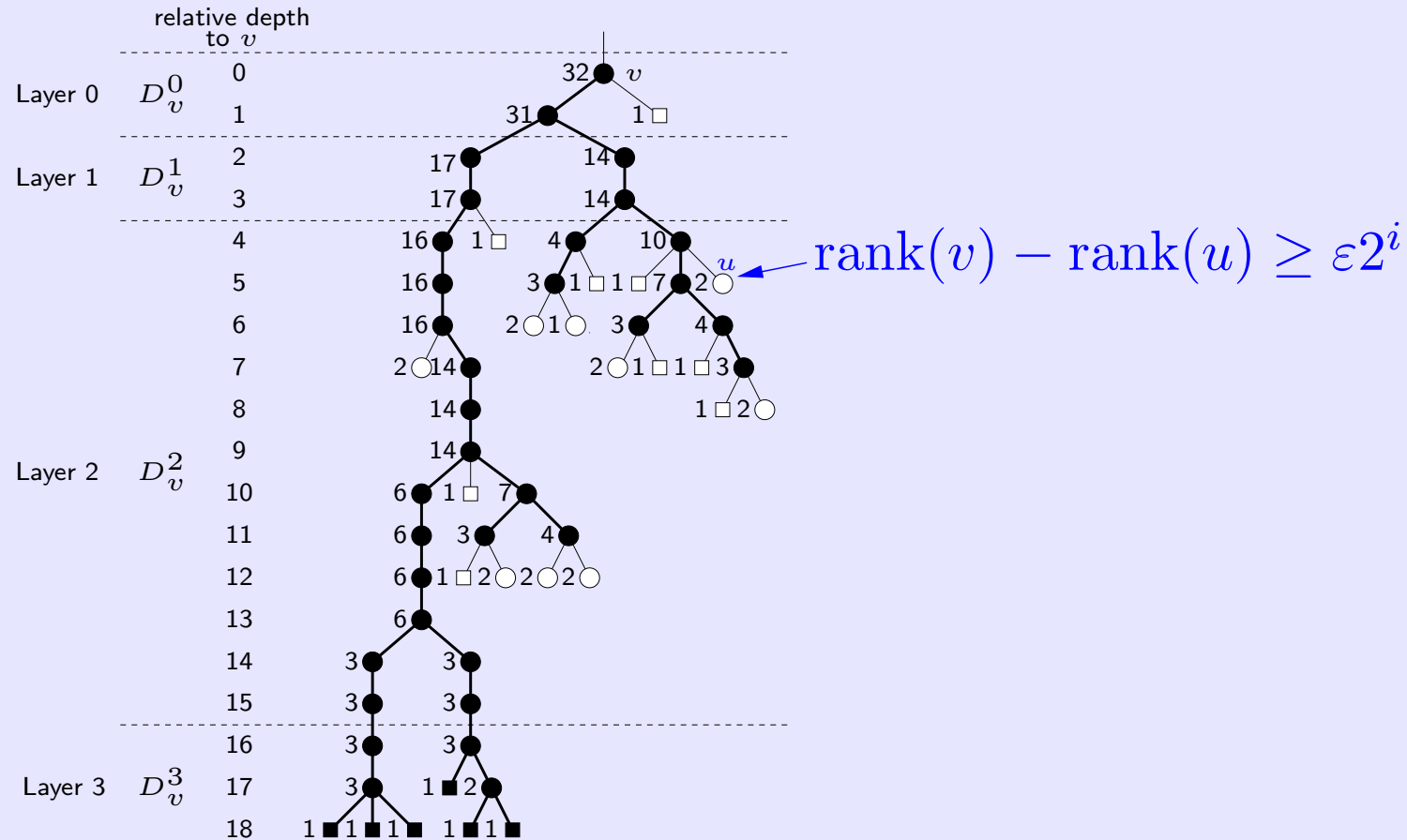


- Partition input trie T into **components** (generalization of heavy paths)
- T' = collapse components in T into high degree nodes and replace by weight balanced trees
- Apply van Emde Boas layout out to T'

Search: $O(\log_B n)$ I/O — **ignoring searching inside components**



Decomposition into Components



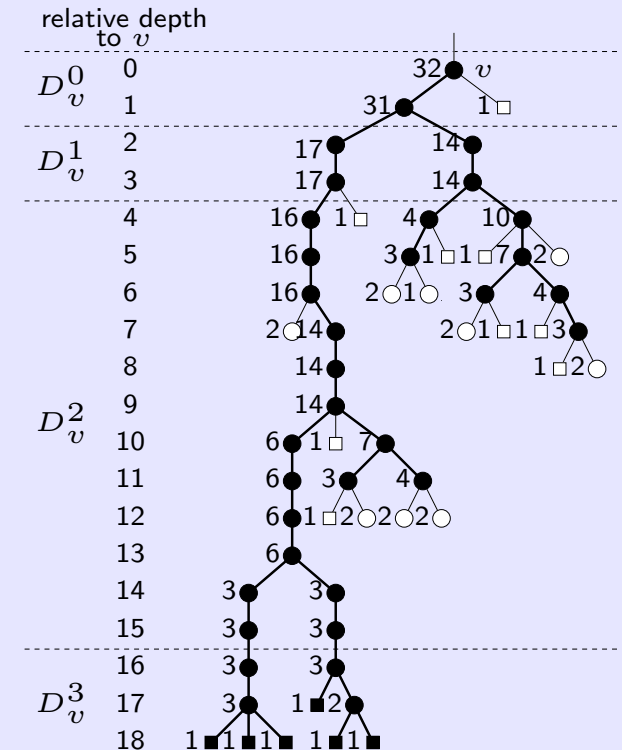
$$D_v^0 = \{u \in T_v \mid \text{rank}(u) = \text{rank}(v) \wedge \text{depth}(u) - \text{depth}(v) < 2^{2^0}\}$$

$$D_v^i = \{u \in T_v \mid \text{rank}(v) - \text{rank}(u) < \epsilon 2^i \wedge 2^{2^{i-1}} \leq \text{depth}(u) - \text{depth}(v) < 2^{2^i}\}$$



Storing and Searching Components

- Store each layer D_v^i separately
- Make a giraf-decomposition of D_v^i
- For D_v^i have a blind trie of size $O(2^{\varepsilon 2^i})$ (using BFS layout) to select the right giraffe-tree
- **Search:** D_v^i search the blind trie + search in one giraffe-tree
- Distribute $D_v^0, D_v^1, D_v^2, \dots$ in the van Emde Boas layout of T'
- **Analysis:**
 - Search in blind trie for D_v^{i+1} (**lookahead**) dominated by the matched characters in D_v^i — requires $M \geq B^{2+\delta}$
 - Space in van Emde Boas layout for a subtree of size k becomes $O(k^3)$



Cache-Oblivious Tries

There exists a cache-oblivious trie supporting prefix queries in

$$O(\log_B |n| + |P|/B) \text{ I/Os,}$$

where P is the query string, and n is the number of leaves in the trie.

It can be constructed in $O(\text{Sort}(N))$ time, where N is the total number of characters in the input.

The space required is $O(N)$.

The structure assumes $M \geq B^{2+\delta}$.



Final Remarks

- Lookahead in the query string is crucial (both cache-aware and cache-oblivious)
- A **giraffe cover** is a simple construction allowing topdown path traversals in a tree using $O(|P|/B)$ I/Os





The End